Documentazione del progetto *Rubrica Contatti*

Documento contenente la progettazione del progetto Rubrica Contatti.

0612707416 - Gioia Iannuzzi
0612708273 - Luigi Montonetti
0612707862 - Arianna Paletta
0612708294 - Debora Villano

Anno Accademico 2024/2025

Indice

1 Introduzione

1.1 Propositi

Il proposito di questo documento è specificare il progetto del sistema software *Rubrica Telefonica*, con l'obiettivo di guidarne la progettazione, realizzazione e validazione. Il documento è redatto seguendo le linee guida stabilite nello standard *IEEE Std 830-1993* (IEEE Recommended Practice for Software Requirements Specifications), per garantire che i requisiti siano chiari, completi e facilmente comprensibili da tutti gli stakeholder coinvolti nel progetto.

1.2 Scopo

Il prodotto software descritto in questo documento è una **rubrica telefonica**, progettata per la gestione ed organizzazione di numeri telefonici. L'applicazione sarà progettata per funzionare su sistemi operativi quali: Windows, macOS e Linux, con funzionalità di ricerca, modifica e organizzazione.

1.3 Pubblico Destinatario

Il pubblico destinatario di questo documento include le persone coinvolte nella progettazione, sviluppo, utilizzo e manutenzione del sistema:

- **Sviluppatori:** Responsabili della progettazione, implementazione e manutenzione del software, che utilizzeranno il documento per comprendere i requisiti, le specifiche tecniche del sistema e il flusso di progettazione corretto da seguire.
- **Tester:** I professionisti che verificheranno il corretto funzionamento del sistema, utilizzando questo documento come riferimento.
- **Utenti Finali:** Le persone che utilizzeranno l'applicazione per gestire i propri contatti. Coloro che potrebbero essere coinvolti nel processo di feedback durante lo sviluppo.
- **Amministratori di Sistema:** Responsabili della gestione dell'infrastruttura, della sicurezza e della manutenzione del sistema, nonché del supporto agli utenti.

1.4 Convenzioni del Documento

Simboli e Terminologia

- SRS: sta per Software Requirements Specifications.
- Acronimi utilizzati per la categorizzazione dei requisiti:
  - IF: Funzionalità Individuale
  - BF: Business Flow
  - DF: Dato e Formato dei Dati
  - UI: Interfaccia Utente
  - IS: Interfaccia con altri Sistemi
  - FC: Ulteriori Vincoli
  - NF: Non Funzionale

Requisiti e Scenari Gli scenari (IF, BF, DF, UI) fungono da indice e permettono di collegare ciascun requisito al relativo flusso o funzionalità. Se un requisito non è collegato ad alcun *business flow*, sarà identificato da un solo pedice (ad esempio, **IS-1** o **FC-1**).

Classificazione delle Priorità La priorità dei requisiti è suddivisa in:

- **Alta**: Requisiti critici per il funzionamento dell'applicazione.
- **Media**: Requisiti utili ma non essenziali, implementabili in una fase successiva.
- **Bassa**: Requisiti opzionali o di minor impatto.

Valutazione del Rischio Tecnico Il rischio tecnico dei requisiti viene classificato come:

- **Basso**: Implementazione semplice e priva di complessità.
- **Medio**: Richiede un moderato sforzo tecnico o risorse specifiche.
- **Alto**: Potrebbe causare problemi significativi o richiedere tecnologie avanzate.

Acronimi Utilizzati

- **GUI** (*Graphical User Interface*): Interfaccia Grafica Utente.
- **CSV** (*Comma-Separated Values*): Formato di file utilizzato per memorizzare dati tabellari in formato testuale, separati da virgole.

Casi d'Uso La spiegazione dei casi d'uso segue il modello strutturato riportato di seguito:

- **Nome del Requisito**: Titolo identificativo.
- **Obiettivo**: Scopo del caso d'uso.
- **Attori**: Persone o sistemi coinvolti.
- **Pre-condizioni**: Requisiti da soddisfare prima di eseguire il caso d'uso.
- **Post-condizioni**: Risultato atteso dopo l'esecuzione del caso d'uso.
- **Input**: Dati o informazioni necessarie per l'esecuzione.
- **Flusso Normale**: Sequenza di eventi standard.
- **Flusso Alternativo**: Varianti o eccezioni al flusso normale.

2 Descrizione Generale

2.1 Prospettiva del Prodotto

L'applicazione che si vuole realizzare è un sistema software autonomo, indipendente da prodotti esterni. Tuttavia, è in grado di integrarsi con questi, garantendo una gestione sicura e accessibile dei dati degli utenti.

2.2 Funzionalità del Prodotto

Le principali funzionalità del prodotto includono:

- **Creazione, Modifica ed Eliminazione dei Contatti:** Consente agli utenti di mantenere una lista aggiornata dei propri contatti.
  Dando la possibilità di inserire nuovi contatti nella propria rubrica telefonica, di modificarne o eliminarne alcuni tra quelli già presenti. Sono liberi di aggiungere dettagli come il nome completo del contatto, il numero di telefono e l'indirizzo email, oltre a eventuali note personalizzate.
- **Visualizzazione Ordinata:** I contatti sono elencati in ordine alfabetico di cognome e nome per rendere più facile la consultazione.
- **Funzionalità di Ricerca Avanzata:** Permette di trovare i contatti utilizzando il nome o cognome completo o parziale, il numero di telefono oppure l'indirizzo email, tramite una ricerca che non fa distinzione tra maiuscole e minuscole.
- **Gestione dei Duplicati:** Il sistema rileva e gestisce i duplicati di contatti, chiedendo conferma per evitare conflitti.
- **Personalizzazione della Rubrica:** Gli utenti hanno la possibilità di annotare ulteriori dati sui contatti, come date di compleanno, relazioni tra i contatti e altre informazioni.

2.3 Utenti e Caratteristiche

Gli utenti *della Rubrica Telefonica* sono divisi in due categorie:

- **Utenti Finali:** Persone che utilizzano l'applicazione per gestire i propri contatti. Gli utenti finali hanno bisogno di un'interfaccia semplice e intuitiva, che consenta di aggiungere, modificare e consultare rapidamente i contatti.
- **Amministratori:** Gli amministratori sono incaricati della configurazione, gestione e della manutenzione generale dell'applicazione.

Il punto di forza del software è la sua facilità d'uso, che lo rende accessibile a utenti con diverse competenze tecniche, inclusi coloro che non hanno esperienza nell'uso di applicazioni simili.

2.4 Vincoli di Design e Implementazione

Standard Tecnici

Il linguaggio di programmazione utilizzato per il software della rubrica telefonica è **Java 8** con l'ausilio di **JavaFX** per la realizzazione dell'interfaccia utente.

Vincoli

Il sistema deve essere in grado di rispondere entro un secondo alla maggior parte delle richieste effettuate dall'utente, come l'aggiunta, la modifica o la ricerca di un contatto. I file di salvataggio devono essere generati e letti in formati standard come **CSV** o **JSON** per garantire l'interoperabilità con altri software.

Limitazioni sui Dati

Il sistema non permetterà di aggiungere più di **3 numeri di telefono** e più di **3 indirizzi e-mail** per ciascun contatto.

Interfaccia

L'interfaccia utente deve essere pratica e facile da utilizzare anche per utenti non esperti. Comprenderà un design minimalista con pochi pulsanti.

Vincoli per il Rilascio

Il sistema deve essere pronto per il rilascio entro il **15 Dicembre**.

2.5 Assunzioni e Dipendenze

Ambiente di Utilizzo

La rubrica telefonica sarà utilizzabile esclusivamente su **computer** (desktop o laptop) e non è prevista una versione per dispositivi mobili o web.

Competenze Base

Si assume che gli utenti abbiano una conoscenza di base sull'utilizzo di un computer, inclusa la capacità di interagire con un'interfaccia grafica e utilizzare applicazioni di uso comune.

Dati

Si assume che gli utenti inseriscano dati corretti per ogni contatto. Sarà implementata una funzionalità di validazione per i numeri di telefono e gli indirizzi e-mail, al fine di garantire il corretto formato dei dati inseriti.

3 Requisiti Specifici

3.1 Requisiti di Interfaccia Esterna

Disponibilità di un'interfaccia grafica (GUI) Il sistema deve offrire un'interfaccia grafica intuitiva. Questo fa sì che gli utenti possano navigare senza una formazione avanzata.

Accessibilità multi-dispositivo Il sistema deve essere accessibile su vari Sistemi Operativi. Questo garantisce un'esperienza di navigazione a più utenti.

3.2 Requisiti Funzionali

I requisiti funzionali descrivono le operazioni principali che il sistema deve essere in grado di offrire per soddisfare le esigenze degli utenti e raggiungere gli obiettivi. Ogni requisito è identificato da un ID (vedi paragrafo 1.4), e classificato per priorità, descrizione e rischio tecnico. Questo per una facile comprensione da parte del cliente.

ID	**Requisito**	**Priorità**	**Breve descrizione**	**Rischio tecnico**
IF-1.1	Creare Contatti	Alta	Deve essere possibile aggiungere nuovi contatti alla rubrica.	Basso
IF-1.2	Modificare Contatti	Alta	È possibile modificare i dettagli di un contatto esistente.	Basso
IF-1.3	Cancellare Contatti	Alta	È possibile rimuovere contatti dalla rubrica chiedendo una conferma.	Basso
IS-1	Salvare e Caricare Contatti	Alta	Le informazioni della rubrica devono essere salvate su file e caricate da file, mantenendo l'intera rubrica.	Medio
BF-1	Validazione dei campi obbligatori	Alta	È obbligatorio inserire un nome e/o un cognome.	Medio
DF-1	Nome e Cognome	Alta	Ogni contatto deve avere un nome e un cognome, ma almeno uno dei due deve essere presente.	Basso
DF-2	Numeri di Telefono	Alta	Ogni contatto può avere da 0 a 3 numeri di telefono.	Basso
IF-2.1	Ricerca di Contatto	Media	È possibile cercare un contatto inserendo la sottostringa iniziale di nome, cognome, numero o mail.	Medio
UI-1	Aggiunta Note di un Contatto	Bassa	È possibile aggiungere delle note ad un contatto.	Medio
IF-2.2	Visualizzazione Contatti	Alta	È possibile visualizzare una lista dei contatti.	Medio

BF-2	Gestione Duplicati	Media	Il sistema richiede conferma per contatti con lo stesso nome, permettendo di sovrascrivere o creare un nuovo contatto con dettagli aggiuntivi.	Medio
UI-2	Aggiunta Foto Contatto	Bassa	È possibile aggiungere una foto ad un contatto.	Alto
UI-3	Interfaccia Grafica	Alta	Il sistema deve disporre di un'interfaccia grafica.	Basso
UI-4	Tastierino di Aggiunta Contatto	Bassa	È possibile salvare un contatto digitando il numero di telefono da tastierino.	Alto
IF-2.3	Menu Preferiti	Media	L'utente può aggiungere contatti alla lista dei preferiti e accedervi rapidamente.	Medio

Tabella 1: Tabella di Categorizzazione
dei Requisiti Funzionali

3.3 Descrizione dei Casi d'Uso

Creazione Contatto

Campo	**Descrizione**
Obiettivo	L'utente può aggiungere contatti alla rubrica.
Attori partecipanti	Utente
Pre-condizioni	Deve essere accessibile l'interfaccia per la creazione del contatto.
Post-condizioni	Il contatto è salvato nel sistema ed è visibile nella rubrica tramite operazioni di ricerca o visualizzazione.
Input	• Nome • Cognome • Numeri di telefono (massimo 3) • E-mail (massimo 3) • Note (opzionale) • Foto (opzionale)
Flusso normale	1. L'utente accede alla schermata di creazione di un nuovo contatto. 2. L'utente inserisce i dati richiesti (nome, cognome, numero di telefono e/o email). 3. L'utente clicca su *Salva*. 4. Il sistema convalida i dati. 5. Il contatto viene salvato nella memoria. 6. Il sistema conferma che l'operazione è avvenuta con successo.

Flusso alternativo 1	**2a. L'utente non compila tutti i campi richiesti** 2a.1. L'utente prova a salvare il contatto senza compilare i campi richiesti. 2a.2. Il sistema mostra un messaggio di errore indicando che deve essere inserito almeno uno tra *nome* o *cognome*. 2a.3. L'utente può correggere l'errore.
Flusso alternativo 2	**2b. Inserimento di formati non validi nel campo di ricerca** 2b.1. L'utente inserisce dati con formati non validi (ad esempio una stringa di caratteri nel campo *numero di telefono*). 2b.2. Il sistema mostra un messaggio di errore indicando il formato corretto dei campi. 2b.3. L'utente può correggere l'errore.
Flusso alternativo 3	**3a. L'utente prova a salvare un contatto già esistente** 3a.1. Il sistema mostra un messaggio di errore indicando l'eventuale duplicazione. 3a.2. L'utente può aggiornare il contatto già esistente o annullare l'operazione.

Modifica Contatto

Campo	Descrizione
Obiettivo	L'utente può modificare i dettagli di un contatto esistente nella rubrica.
Attori partecipanti	Utente
Pre-condizioni	• La rubrica deve contenere almeno un contatto. • Il contatto da modificare deve essere individuabile tramite ricerca e selezione.
Post-condizioni	• I dati di un contatto vengono aggiornati con nuovi dati forniti dall'utente. • Il contatto aggiornato è visibile nella rubrica.
Input	• Nome • Cognome • Sotto-stringa del nome • Sotto-stringa del cognome • Numero di telefono • E-mail • Dati aggiornati che l'utente vuole modificare

Flusso normale	1. L'utente accede alla rubrica e cerca il contatto da modificare. 2. L'utente seleziona l'opzione modifica. 3. L'utente modifica uno o più campi. 4. L'utente clicca su "Salva". 5. Il sistema convalida i dati. 6. Il contatto viene salvato nella memoria. 7. Il sistema conferma che l'operazione è avvenuta con successo.
Flusso alternativo 1	**1a. Contatto non presente in rubrica:** 1a.1. Il sistema mostra un messaggio di errore indicando che il contatto cercato non è presente.
Flusso alternativo 2	**3a. Modifiche con formati non validi** 3a.1. Il sistema mostra un messaggio di errore indicando il formato corretto dei campi. 3a.2. L'utente può correggere l'errore.
Flusso alternativo 3	**4a. Nessuna modifica effettuata:** 4a.1. Il sistema mostra un messaggio di errore indicando che non è stata effettuata alcuna modifica. 4a.2. L'utente può scegliere se modificare o uscire dall'opzione di modifica.
Flusso alternativo 4	**4b. Contatto duplicato dopo la modifica:** 4b.1. Il sistema mostra un messaggio di errore indicando la duplicazione. 4b.2. L'utente può unire i contatti duplicati o annullare l'operazione.

Eliminazione contatto

Campo	Descrizione
Obiettivo	L'utente può eliminare un contatto esistente dalla rubrica.
Attori partecipanti	Utente
Pre-condizioni	• La rubrica deve contenere almeno un contatto. • Il contatto da eliminare deve essere individuabile tramite ricerca e selezione.
Post-condizioni	• Il contatto selezionato dall'utente viene rimosso dalla rubrica. • Il contatto rimosso non è più visibile.
Input	• Nome • Cognome • Sotto-stringa del nome • Sotto-stringa del cognome • Numero di telefono • E-mail
Flusso normale	1. L'utente accede alla rubrica e cerca il contatto da eliminare. 2. L'utente seleziona l'opzione "Elimina". 3. Il sistema chiede una conferma per l'eliminazione. 4. L'utente conferma l'operazione. 5. Il sistema rimuove il contatto dalla rubrica e dalla memoria. 6. Il sistema conferma che l'operazione è avvenuta con successo.

Flusso alternativo 1	**1a. Contatto non presente in rubrica:** 1a.1. L'utente cerca di eliminare un contatto non presente. 1a.2. Il sistema mostra un messaggio di errore indicando che il contatto non è presente nella rubrica.
Flusso alternativo 2	**1b. Inserimento di formati non validi nel campo ricerca** 1b.1. Il sistema mostra un messaggio di errore e richiede l'inserimento di criteri validi.. 1b.2. L'utente corregge i criteri e riprende il flusso normale.
Flusso alternativo 3	**3a. Annullamento dell'operazione:** 3a.1. L'utente seleziona "Elimina", ma decide di annullare l'operazione. 3a.2. Il sistema non rimuove il contatto.

Salvataggio e Caricamento Contatti

Campo	Descrizione
Obiettivo	L'utente può salvare i contatti della rubrica in un file o database per conservarli in modo sicuro.
Attori partecipanti	Utente
Pre-condizioni	• La rubrica deve contenere almeno un contatto. • Il percorso di salvataggio deve essere accessibile.
Post-condizioni	• I contatti vengono salvati nel formato scelto (es. CSV, JSON, ecc.).
Input	• Rubrica da salvare • Formato in cui salvare la rubrica
Flusso normale	1. L'utente accede alla rubrica. 2. L'utente seleziona l'opzione "Salva contatti". 3. Il sistema chiede il percorso su cui salvare i contatti. 4. L'utente clicca su "Salva". 5. Il sistema esegue il salvataggio. 6. Il sistema conferma che l'operazione è avvenuta con successo.
Flussi alternativo 1	**3a. Il percorso selezionato non è valido.** 3a.1. Il sistema non riesce ad accedere al percorso specificato e mostra un messaggio di errore. 3a.2. L'utente seleziona un nuovo percorso valido. 3a.3. Il sistema riprende il flusso normale e completa il salvataggio.

Flusso alternativo 2	**5a. Si verifica un errore durante il salvataggio (es. mancanza di spazio o inserimento di dati con formati non validi).** 5a1. Il sistema mostra un messaggio di errore indicando la causa dell'impossibilità di salvataggio. 5a2. L'utente può riprovare il salvataggio, scegliendo un percorso alternativo o liberando spazio. 5a3. L'utente può anche decidere di annullare l'operazione.

Ricerca Contatto

Campo	Descrizione
Obiettivo	L'utente può cercare un contatto specifico nella rubrica, basandosi su informazioni come nome, cognome, numero di telefono o indirizzo e-mail.
Attori partecipanti	Utente
Pre-condizioni	Deve essere accessibile l'interfaccia per la ricerca dei contatti.
Post-condizioni	• Vengono visualizzati i contatti corrispondenti ai criteri di ricerca.
Input	• Nome • Cognome • Numero di telefono • E-mail
Flusso normale	1. L'utente accede alla schermata di ricerca dei contatti. 2. L'utente inserisce le informazioni. 3. L'utente clicca su "Cerca". 4. Il sistema restituisce i risultati della ricerca. 5. L'utente può visualizzare e selezionare uno dei risultati.
Flusso alternativo 1	**1b. L'utente inserisce formati non validi nel campo ricerca** 1b.1. Il sistema mostra un messaggio di errore e richiede l'inserimento di criteri validi.. 1b.2. L'utente corregge i criteri e riprende il flusso normale.
Flusso alternativo 2	**3b. L'utente inserisce informazioni non presenti nella rubrica.** 3b.1. Il sistema mostra un messaggio di errore indicando che il contatto non è presente in rubrica.

Gestione del *Menù Preferiti*

Campo	Descrizione
Obiettivo	L'utente può aggiungere, visualizzare e gestire i contatti in un menu per un accesso rapido.
Attori partecipanti	Utente
Pre-condizioni	Deve essere accessibile l'interfaccia per la gestione dei preferiti.
Post-condizioni	• I contatti selezionati sono visibili nel menu preferiti. • L'utente può accedere rapidamente ai contatti preferiti.
Input	• Contatti da aggiungere ai preferiti
Flusso normale	1. L'utente seleziona o cerca un contatto da aggiungere nel menu preferiti. 2. L'utente clicca su Aggiungi ai preferiti. 3. Il sistema convalida i dati. 4. Il contatto viene salvato nel menu preferiti. 5. L'utente può visualizzare l'elemento nel menu preferiti.
Flusso alternativo 1	**2a. L'utente cerca di aggiungere un contatto già presente nei preferiti.** 2a.1. Il sistema mostra un messaggio di errore indicando che l'elemento è già presente.
Flusso alternativo 2	**2b. L'utente rimuove un contatto dai preferiti.** 2b.1. L'utente seleziona un contatto e clicca su *Rimuovi dai preferiti*. 2b.2. Il sistema rimuove l'elemento dal menu preferiti.

Flusso alternativo 3	**3b. Inserimento di formati non validi nel campo ricerca** 3b.1. Il sistema mostra un messaggio di errore e richiede l'inserimento di criteri validi.. 3b.2. L'utente corregge i criteri e riprende il flusso normale.

Visualizzazione Contatto

Campo	Descrizione
Obiettivo	L'utente può visualizzare i dettagli di un contatto presente nella rubrica.
Attori partecipanti	Utente
Pre-condizioni	• La rubrica deve contenere almeno un contatto. • Il contatto da visualizzare deve essere individuabile tramite ricerca o selezione.
Post-condizioni	• L'utente può consultare tutte le informazioni associate al contatto selezionato.
Input	• Nome • Sottostringa del nome • Cognome • Sottostringa del cognome • Numero di telefono • E-mail

Flusso normale	1. L'utente inserisce i criteri di ricerca (es. nome, cognome o altro identificatore). 2. Il sistema ricerca e presenta l'elenco dei contatti corrispondenti ai criteri inseriti. 3. L'utente seleziona il contatto desiderato dall'elenco. 4. Il sistema mostra i dettagli completi del contatto, inclusi: • Nome • Sottostringa del nome • Cognome • Sottostringa del cognome • Numeri di telefono • Indirizzi E-mail • Note (se presenti) • Foto (se presente)
Flussi alternativi	**1a. L'utente cerca un contatto non presente.** 1a.1. Il sistema mostra un messaggio di errore indicando che il contatto non è presente nella rubrica.
Flusso alternativo 2	**1b. Inserimento di criteri di ricerca non validi** 1b.1. Il sistema mostra un messaggio di errore e richiede l'inserimento di criteri validi.. 1b.2. L'utente corregge i criteri e riprende il flusso normale.

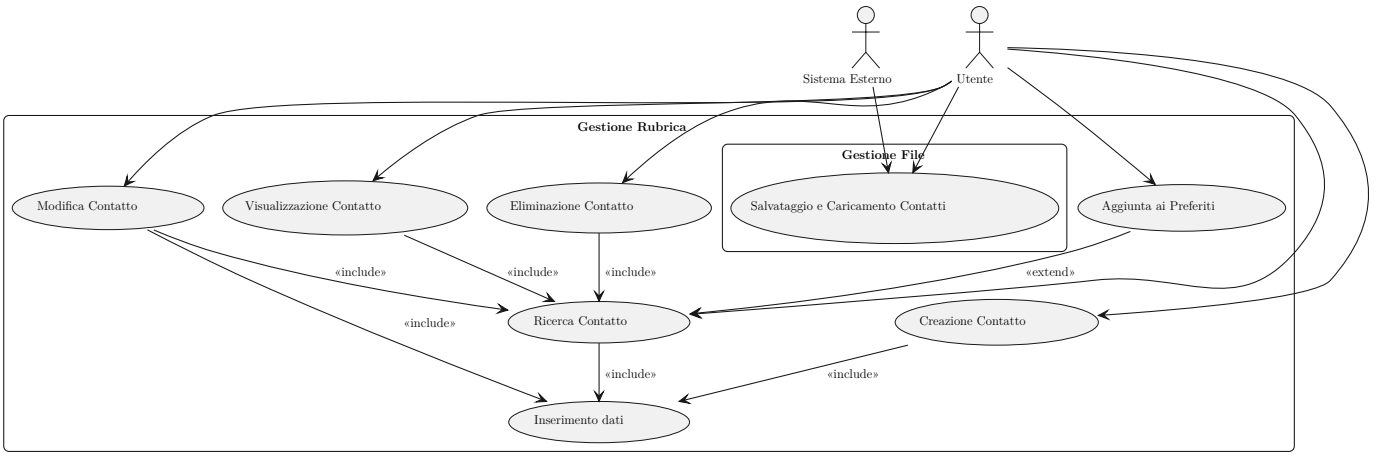

Figura 1: Diagramma UML dei Casi d'Uso
Realizzato con PlantUML

4 Requisiti Non Funzionali e Attributi di Qualità

In questa sezione vengono descritti i requisiti non funzionali e gli attributi di qualità che il sistema deve soddisfare per garantire efficienza, sicurezza e un'esperienza utente ottimale. Questi requisiti definiscono le prestazioni attese, le misure di sicurezza e gli attributi di qualità che influiscono sulla manutenibilità, modularità ed efficacia complessiva del sistema. Tali aspetti sono fondamentali per assicurare non solo il corretto funzionamento del software, ma anche la sua capacità di adattarsi e rispondere alle esigenze degli utenti e degli sviluppatori nel tempo.

4.1 Requisiti di Prestazione

- **Ricerca di un contatto:** La ricerca di un contatto deve avvenire in meno di un secondo.
- **Modifica di un contatto:** La modifica di un contatto deve avvenire in maniera rapida, in meno di un secondo.
- **Creazione di un contatto:** La creazione di un contatto deve avvenire in meno di un secondo.
- **Velocità caricamento:** L'interfaccia grafica dell'utente deve aprirsi in maniera rapida senza scatti.

4.2 Requisiti di Sicurezza e Protezione

- **Protezione dei dati:** I dati della rubrica devono essere protetti contro accessi non autorizzati utilizzando un sistema di password o crittografia.
- **Interazione:** L'interazione con le funzionalità dell'interfaccia grafica deve essere fluida e non causare ritardi.
- **Gestione della memoria:** I dati sono salvati in un file JSON.

Requisito	**Priorità**	**Categoria**	**Breve descrizione**	**Rischio tecnico**
L'interfaccia GUI	Alta	Usabilità	L'interfaccia utente deve essere visiva e basata su elementi grafici, facilitando l'interazione.	Basso
Formato numeri	Bassa	Affidabilità	I numeri di telefono devono rispettare un formato internazionale standard (es. +39 123 456 789).	Medio
Formato email	Bassa	Affidabilità	Gli indirizzi email devono essere validi secondo le specifiche RFC 5322.	Medio
Sicurezza dei Dati	Alta	Sicurezza	I dati della rubrica devono essere protetti contro accessi non autorizzati.	Basso
Limite Preferiti	Bassa	Usabilità	Il sistema deve permettere un massimo di 20 contatti nella lista dei preferiti.	Medio
Supporto Multilingua	Bassa	Accessibilità	L'interfaccia deve essere disponibile in almeno due lingue (es. italiano e inglese) e facilmente estensibile ad altre lingue.	Alto
Comportamento di Ricerca	Alta	Usabilità	La ricerca deve essere case-insensitive e supportare caratteri speciali.	Basso
Backup Automatico	Bassa	Affidabilità	Deve essere possibile configurare un backup automatico giornaliero o settimanale dei dati della rubrica.	Medio
Prestazioni Applicative	Alta	Prestazioni	L'applicazione deve rispondere alle operazioni comuni (es. ricerca, aggiunta, modifica) entro 1 secondo per il 95% dei casi.	Medio

Tabella 9: Categorizzazione dei Requisiti Non Funzionali

4.3 Attributi di Qualità

4.3.1 QA Interni

- **Manutenibilità:** Il software della rubrica è scritto in un linguaggio chiaro (Java) e ben documentato, con commenti, in questo modo sarà più facile per uno sviluppatore esterno risolvere bug.

- **Modularità:** Il codice è scritto in maniera tale da avere una modularità alta, ci sono diverse classi con diversi metodi per far sì che ogni funzionalità sia a sé. Utilizzando questo approccio è più facile cambiare un metodo nella sua singolarità.

- **Efficienza del codice:** È stato utilizzato un algoritmo di ricerca basato sulle liste in modo da rendere più efficiente la ricerca di un contatto all'interno della rubrica

- **Riusabilità:** Il software è diviso in diverse classi, questo rende più facile la riusabilità di alcune parti del codice in contesti diversi.

4.3.2 QA Esterni

- **Usabilità:** Un utente può cercare un contatto facilmente e in maniera intuitiva cercando solamente il suo nome e/o cognome o uno dei numeri o delle email.

- **Prestazioni:** L'aggiunta, la modifica e la ricerca di un contatto sono tutte funzionalità della rubrica che impiegheranno meno di un secondo per compiere la loro azione.

- **Sicurezza (Safety):** I dati verranno salvati su un file JSON in modo da non creare danni per la gestione delle informazioni dei contatti nella rubrica.

5 Stato Dei Requisiti

ID	**Priorità**	**Stato del requisito**	**Rischio tecnico**
IF-1.1	Alta	Accettato per questa release	Basso
IF-1.2	Alta	Accettato per questa release	Basso
IF-1.3	Alta	Accettato per questa release	Basso
IS-1	Alta	Accettato per questa release	Medio
BF-1	Alta	Accettato per questa release	Medio
BF-2	Media	Accettato per questa release	Medio
DF-1	Alta	Accettato per questa release	Basso
DF-2	Alta	Accettato per questa release	Basso
IF-2.1	Media	Accettato per questa release	Medio
IF-2.2	Alta	Accettato per questa release	Medio
UI-1	Bassa	Accettato per questa release	Medio
UI-2	Bassa	Accettato per questa release	Alto
UI-3	Alta	Accettato per questa release	Basso
UI-4	Bassa	Accettato per questa release	Alto
IF-2.3	Media	Accettato per questa release	Medio

Tabella 10: Tabella di Stato
dei Requisiti Funzionali

Requisito	**Priorità**	**Stato del requisito**	**Rischio tecnico**
L'interfaccia GUI	Alta	Accettato per questa release	Basso
Formato numeri	Bassa	Accettato per questa release	Medio
Formato email	Bassa	Accettato per questa release	Medio
Sicurezza dei Dati	Alta	Accettato per questa release	Basso
Limite Preferiti	Bassa	Rimandato per questa release	Medio
Supporto Multilingua	Bassa	Rimandato per questa release	Alto
Comportamento di Ricerca	Alta	Accettato per questa release	Basso
Backup Automatico	Bassa	Rimandato per questa release	Medio
Prestazioni Applicative	Alta	Accettato per questa release.	Medio

Tabella 11: Tabella di Stato
dei Requisiti non Funzionali

6 Design Architetturale

L'architettura del sistema segue il pattern MVC (Model-View-Controller), garantendo la separazione delle responsabilità e migliorando la manutenibilità, la scalabilità e la riutilizzabilità del codice.
Per la strutturazione del sistema, si è adottato un approccio MVC, un pattern architetturale che consente di suddividere il sistema in tre componenti principali:

- **Model**: Gestisce i dati e la logica di business.
- **View**: Rappresenta l'interfaccia utente.
- **Controller**: Gestisce gli input dell'utente e aggiorna il modello.

Tale approccio permette una netta separazione delle responsabilità, migliora la riutilizzabilità del codice e incrementa la manutenibilità e scalabilità del sistema.

6.1 View

Ogni GUI è sviluppata tramite il file FXML, mentre il design visivo è definito nei corrispondenti CSS. Di seguito vengono elencate tutte le **View**

View (FXML)	Descrizione	Controller Associato
`InterfacciaUtente.fxml`	Gestione rubrica principale	`InterfacciaUtenteController.java`.
`MenuPreferiti.fxml`	Gestione dei contatti preferiti	`MenuPreferitiController.java`
`InterfacciaAggiungiModifica.fxml`	Aggiunta e modifica contatti	`InterfacciaAggiungiModificaController.java`

Tabella 12: Elenco delle View del Sistema

6.2 Controller

I controller gestiscono l'interazione tra l'utente (input), la View, e il Model. Di seguito la lista dei Controller adottati:

Controller	Descrizione	View Associata
`InterfacciaUtenteController.java`	Gestisce l'interfaccia principale della rubrica, inclusa la visualizzazione dati.	`InterfacciaUtente.fxml`.
`MenuPreferitiController.java`	Gestisce la lista dei contatti preferiti, inclusi aggiornamenti e rimozioni.	`MenuPreferiti.fxml`
`InterfacciaAggiungiModificaControl`	Aggiunta e modifica contatti	`InterfacciaAggiungiModificaController.fxml`

Tabella 13: Elenco dei Controller del Sistema

6.3 Model

Il Model rappresenta i dati e la logica di business dell'applicazione. Di seguito vengono elencati i componenti principali del Model:

Classe Model	Descrizione
`Contatto`	Rappresenta un singolo contatto con attributi come nome, cognome, telefono, email e note.
`Rubrica`	Gestisce la lista principale dei contatti e la lista dei preferiti.
`SalvaCaricaRubrica`	Si occupa della persistenza dei dati della rubrica su file JSON o CSV.
`SalvaCaricaPreferiti`	Si occupa della persistenza dei contatti preferiti su file JSON.
`ContattoValidator`	Valida i campi inseriti (nome, email, telefono) e rileva eventuali duplicati.

Tabella 14: Elenco del Model del Sistema

6.3.1 Funzionalità chiave del Model

- **Persistenza dei dati**: Utilizzo di file JSON/CSV per il salvataggio e il caricamento dei contatti.
- **Validazione**: Controllo di errori nei campi inseriti e prevenzione dei duplicati.
- **Gestione dati**: Aggiunta, modifica, eliminazione e ricerca dei contatti.

7 Diagrammi di sequenza

7.1 Creazione Contatto

L'utente avvia l'azione dalla finestra principale, che apre una nuova finestra di creazione. Il controller raccoglie i dati (nome, cognome, numeri, email, nota). Se validi, il contatto viene aggiunto, ordinato e salvato. Errori come campi vuoti, numeri non validi o campi duplicati generano un messaggio di errore.

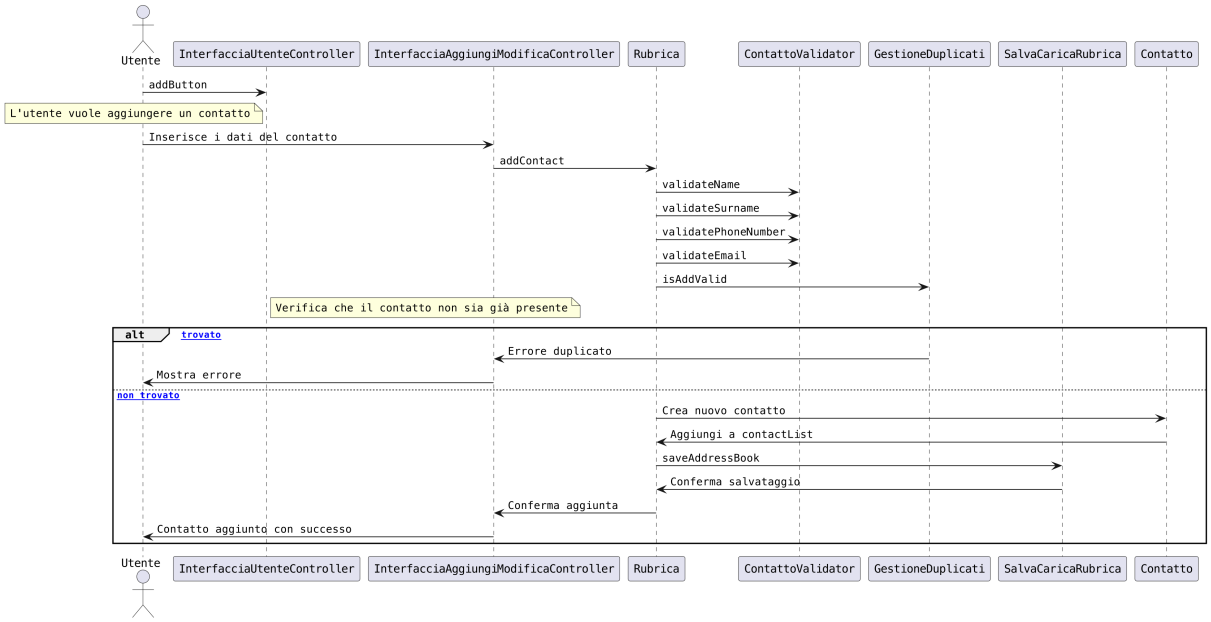

Figura 2: Creazione di un Contatto

7.2 Modifica Contatto

Quando un contatto viene modificato tramite l'interfaccia utente, il controller passa i dati aggiornati alla rubrica, che valida le informazioni e verifica eventuali duplicati. Se la modifica è valida, il contatto viene sincronizzato sia nella lista principale che in quella dei preferiti, se presente. Infine, le liste vengono salvate. In caso di dati non validi o duplicati, viene mostrato un messaggio di errore all'utente.

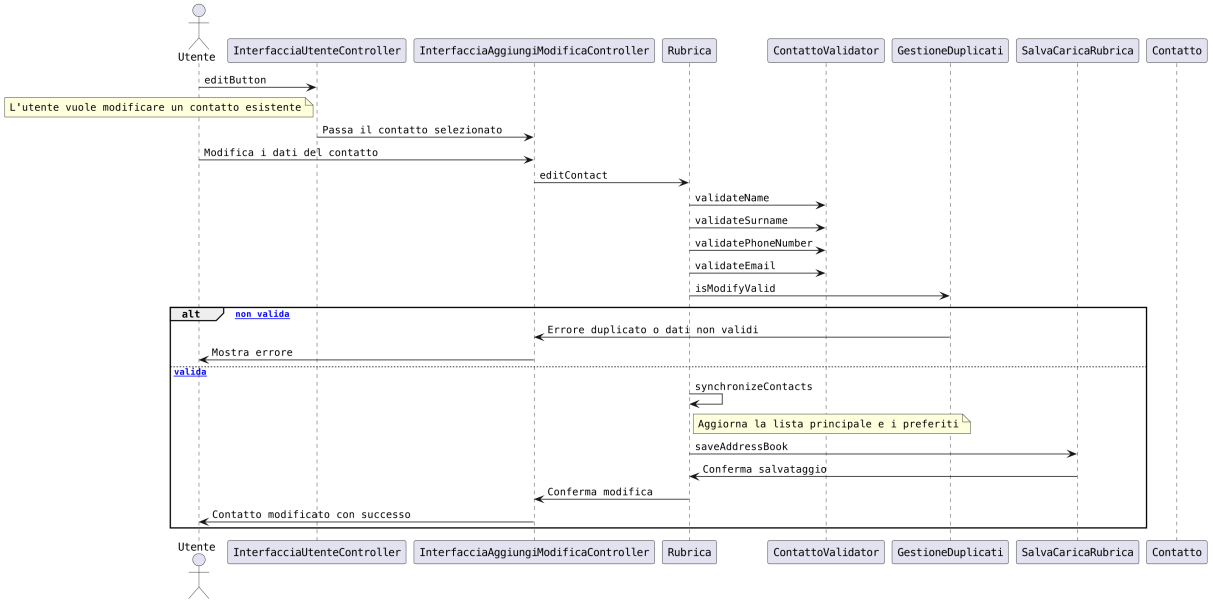

Figura 3: Modifica di un Contatto

7.3 Eliminazione Contatto

Quando l'utente seleziona un contatto e conferma l'azione, il controller rimuove il contatto dalla lista principale e, se presente, anche dalla lista dei preferiti. Infine, entrambe le liste aggiornate vengono salvate.

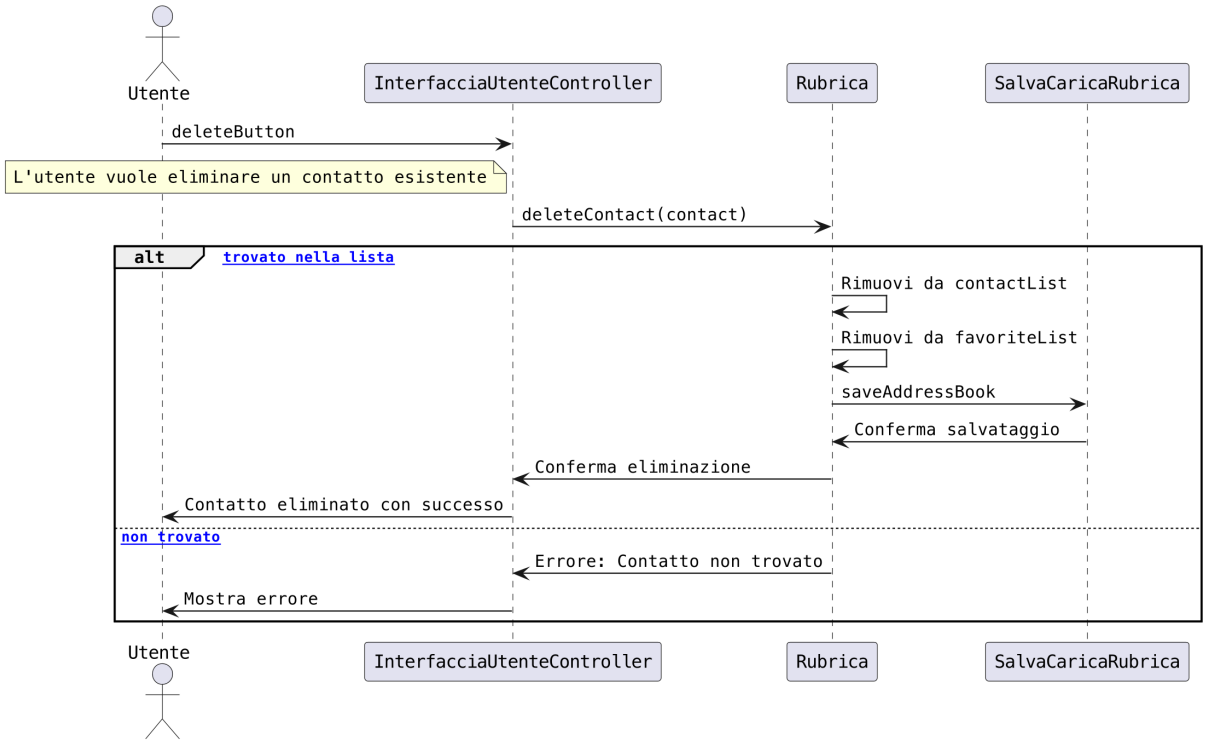

Figura 4: Eliminazione di un Contatto

7.4 Gestione del Menu Preferiti

Il diagramma mostra le funzionalità del Menu Preferiti e la loro gestione interna.

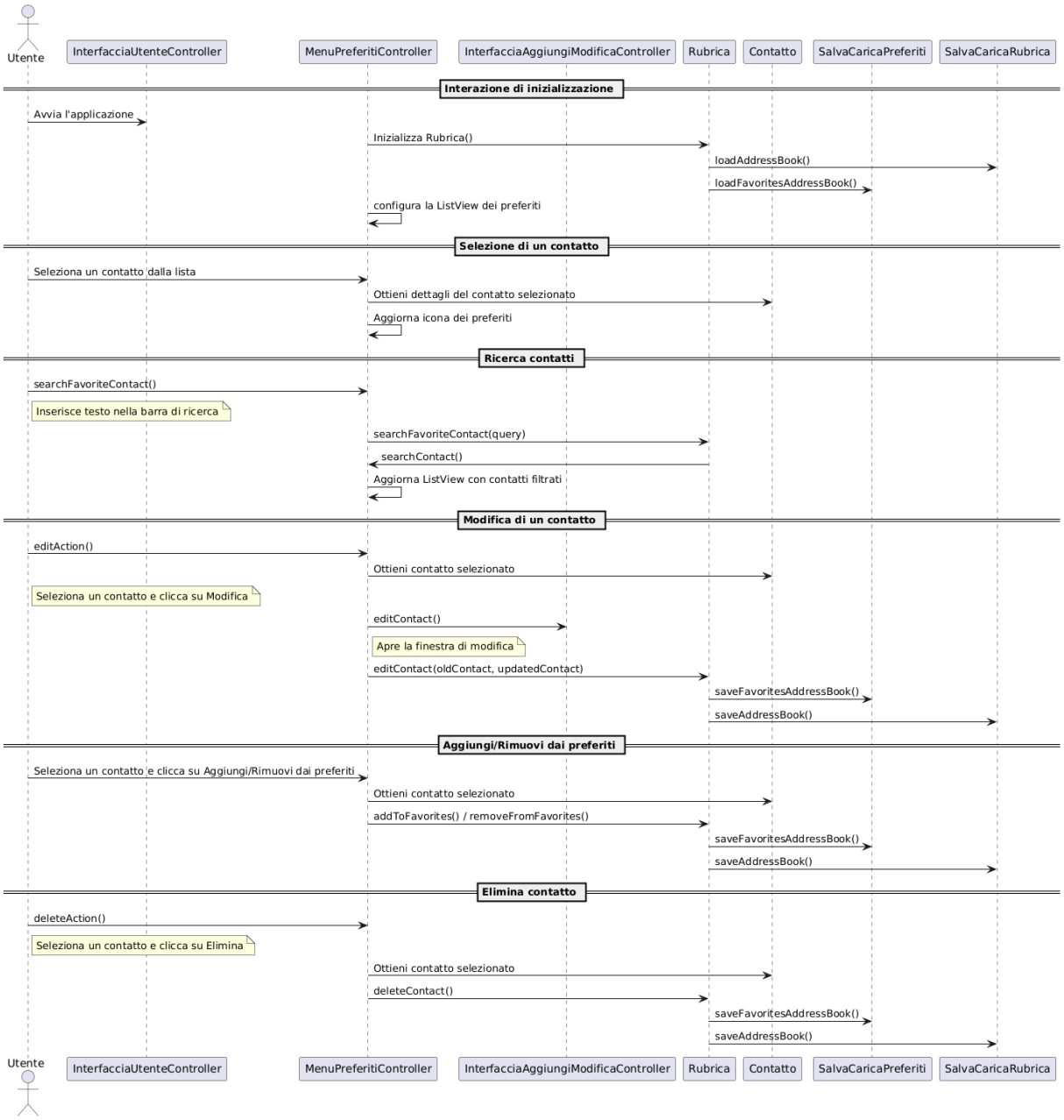

Figura 5: Gestione del Menù Preferiti

7.5 Ricerca Contatto

L'utente inserisce il testo per la ricerca. L'Interfaccia Utente passa il parametro di ricerca al modello. La Rubrica filtra la lista dei contatti confrontando il parametro con i campi dei contatti. Se vengono trovati contatti, la lista filtrata viene restituita e visualizzata. Se non viene trovato nulla, viene mostrato un messaggio di errore.

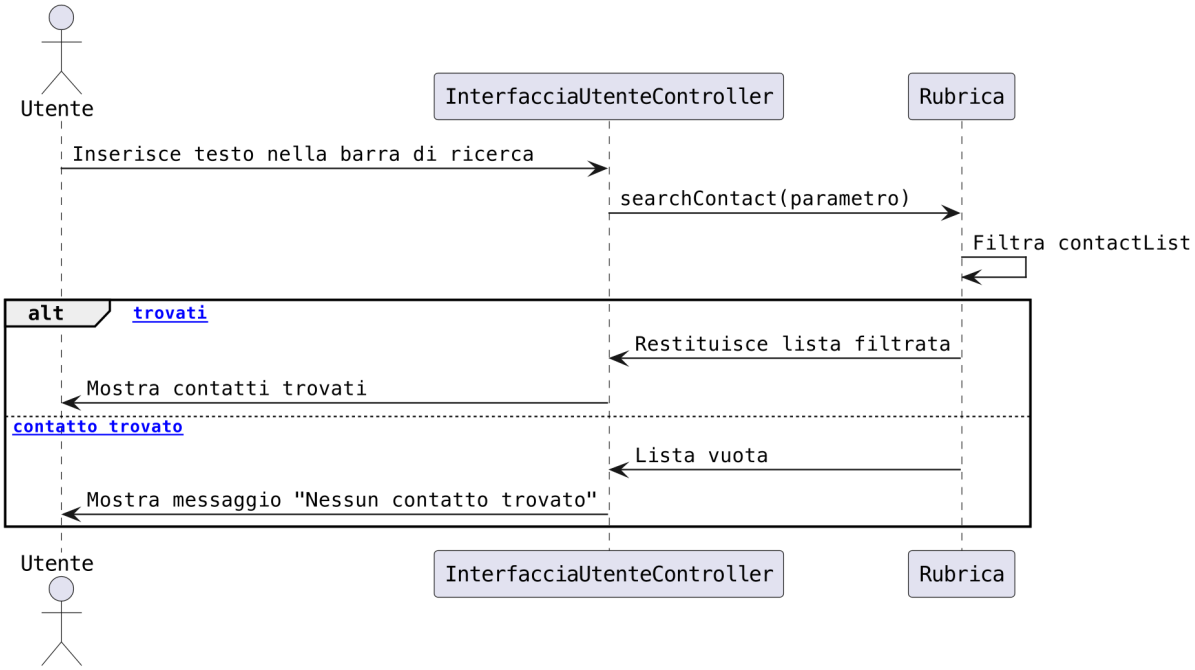

Figura 6: Ricerca Contatto

8 Diagramma delle Classi

Il diagramma delle classi mostrato nella figura sottostante illustra le principali classi e le loro relazioni.

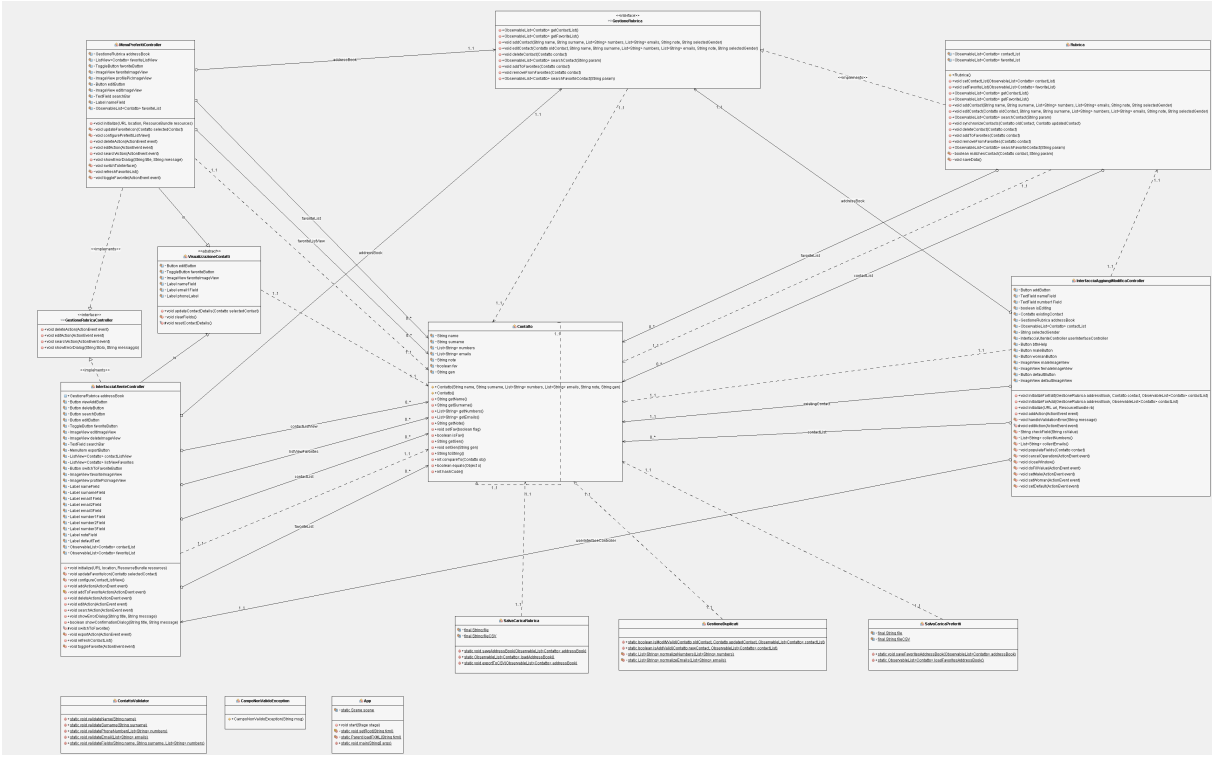

Figura 7: Diagramma delle Classi di Alto Livello

8.1 Descrizione delle classi

- **Classe `App`**: Rappresenta l'applicazione principale. Gestisce l'avvio e la terminazione dell'applicazione
- **Classe `Contatto`**: Rappresenta un contatto con un nome, un cognome, da uno a tre numeri di telefono, da una a tre email, delle note, un campo 'preferito' e il genere.
- **Interfaccia `GestioneRubrica`** : Definisce i metodi necessari per la gestione dei contatti nella rubrica principale e in quella dei preferiti, come l'aggiunta, la modifica, l'eliminazione e la ricerca dei contatti.
- **Interfaccia `GestioneRubricaController`**: Definisce i metodi che devono essere implementati dai controller per gestire le azioni comuni nella rubrica, come eliminare, modificare, cercare contatti e mostrare errori.
- **Classe `InterfacciaAggiungiModifica`**: Gestisce gli eventi e le interazioni dell'utente per aggiungere e modificare contatti. Include l'aggiornamento della rubrica.
- **Classe `InterfacciaUtenteController`**: Controlla la gestione della GUI e fornisce funzionalità per aggiungere, modificare, eliminare e cercare contatti. Inoltre permette l'aggiunta o la rimozione dei contatti dai preferiti

- **Classe `MenuPreferitiController`**: Gestisce la visualizzazione e le operazioni sui contatti preferiti, tra cui aggiungere, rimuovere, modificare e cercare i contatti nella lista dei preferiti. Fornisce inoltre funzionalità di gestione della GUI, come l'aggiornamento dei dettagli del contatto selezionato e la sincronizzazione con la rubrica principale.
- **Classe `Rubrica`** : Gestisce due `ObservableList`: di contatti e di preferiti, consentendo operazioni come aggiunta, modifica, eliminazione, e ricerca di contatti. Include la gestione dei duplicati, la validazione dei campi, la sincronizzazione tra contatti e preferiti, e il salvataggio/caricamento dei dati.
- **Classe `SalvaCaricaRubrica`**: Questa classe fornisce i metodi per salvare una lista di contatti in un file JSON o per caricarla da un file JSON esistente. Inoltre si occupa di esportarli in un file CSV.
- **Classe `SalvaCaricaPreferiti`**: Si occupa del salvataggio e del caricamento della rubrica preferiti in un file JSON garantendo che i dati siano sempre aggiornati.
- **Classe `ContattoValidator`**: Verifica se i dati inseriti sono in un formato valido.
- **Classe `CampoNonValidoException`**: Questa classe estende l'eccezione `Exception` per segnalare errori relativi a campi non validi, come un nome o un cognome.
- **Classe `GestioneDuplicati`**: Classe che verifica se l'aggiunta o la modifica di un contatto generino duplicati.

9 Diagramma delle Attività

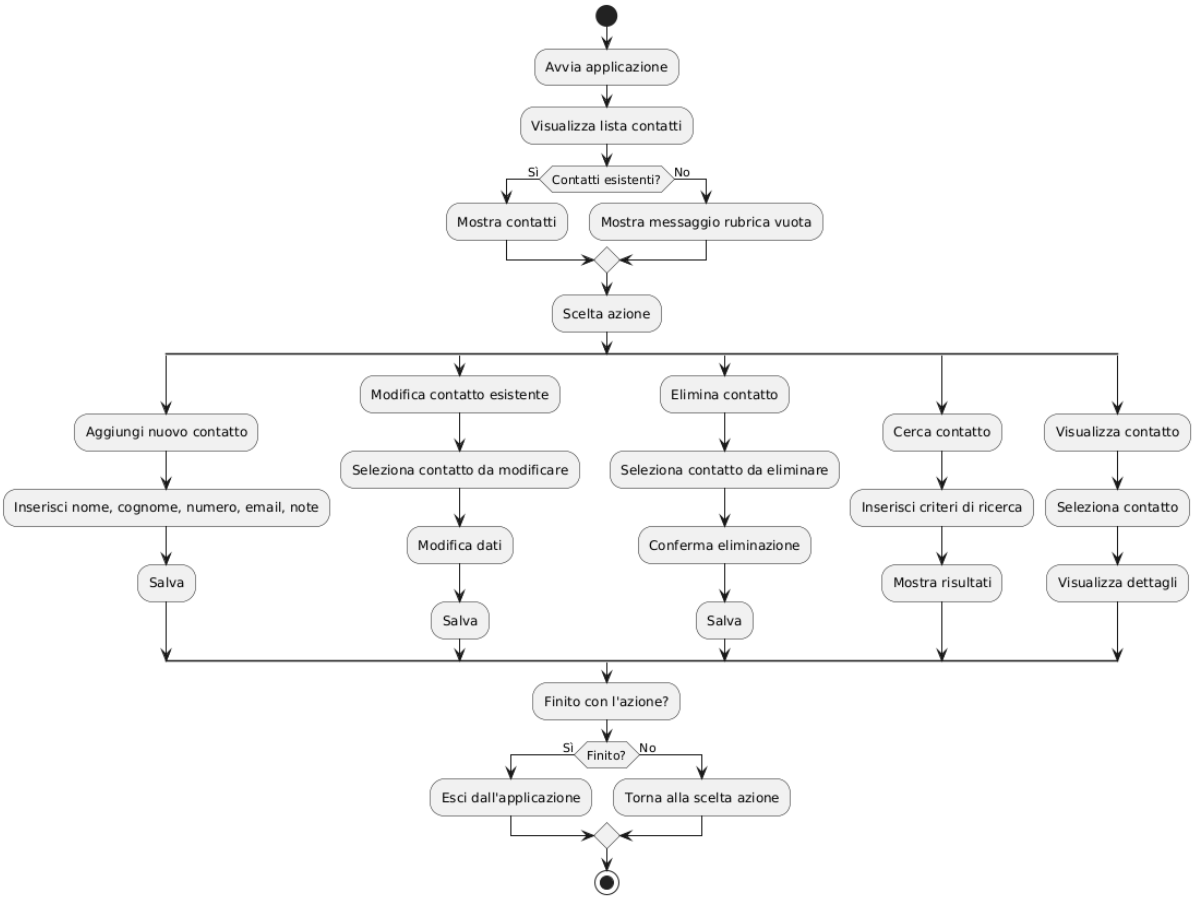

Figura 8: Diagramma delle Attività

10 Design dell' Interfaccia Utente

Il design dell'interfaccia utente è stato sviluppato sulla base dei casi d'uso identificati, garantendo una navigazione chiara e fluida tra le diverse schermate dell'applicazione. L'obiettivo è creare un'esperienza utente semplice e intuitiva, che permetta di gestire la rubrica in modo efficace. Di seguito vengono descritti i dettagli di ciascuna interfaccia e le modalità di navigazione tra di esse:

- **Schermata principale** (*InterfacciaUtenteController*)
  La schermata principale mostra una lista di contatti. L'utente può aggiungere nuovi contatti, modificare o eliminare quelli esistenti. Da questa schermata è possibile:
  - **Aggiungere un nuovo contatto**: L'utente può cliccare su un pulsante per accedere alla schermata di aggiunta di un contatto: *InterfacciaAggiungiController*.
  - **Accedere ai preferiti**: Un altro pulsante consente di visualizzare i contatti preferiti tramite la schermata *MenuPreferitiController*.
- **Schermata *Aggiunta Contatto*** (*InterfacciaAggiungiController*)
  Questa schermata permette di inserire nuovi contatti nella rubrica. Include campi per il nome, il cognome, il numero di telefono, l'email e le note. Da questa schermata è possibile:
  - **Confermare e Salvare la creazione di un contatto**: Dopo aver completato il modulo, l'utente può cliccare su un pulsante per aggiungere il contatto alla rubrica. L'operazione aggiorna la lista principale dei contatti e ritorna alla schermata principale.
  - **Annullare la creazione di un contatto**: Se l'utente non vuole aggiungere il contatto, può annullare l'operazione e tornare alla schermata principale senza modifiche.
- **Schermata *Menù Preferiti*** (*MenuPreferitiController*)
  La schermata dei preferiti mostra i contatti contrassegnati come preferiti. Gli utenti possono visualizzare, aggiungere o rimuovere contatti dai preferiti. Da questa schermata è possibile:
  - **Selezionare contatti dalla rubrica**: L'utente può cliccare su un pulsante che apre una finestra pop-up *SelezionaContattiDaRubricaController* per selezionare i contatti da aggiungere ai preferiti.
  - **Tornare alla schermata principale**: Un altro pulsante consente di tornare alla schermata principale senza modificare i preferiti.

11 Analisi di Coesione e Accoppiamento dei Moduli del Sistema

L'analisi dei moduli sviluppati è presentata nella tabella seguente, con particolare attenzione al livello di coesione interna e di accoppiamento esterno, evidenziando i punti di forza e le aree di miglioramento.

Modulo	Livello di Coesione	Descrizione
`Contatto`	Funzionale	Rappresenta l'entità "Contatto" con tutte le sue proprietà (nome, cognome, numeri, email, note, genere).
`Rubrica`	Funzionale	Implementa tutte le operazioni relative alla gestione della rubrica (aggiunta, modifica, eliminazione, ricerca, gestione dei preferiti).
`InterfacciaUtenteController`	Funzionale	Gestisce tutte le interazioni con l'utente attraverso l'interfaccia grafica.
`MenuPreferitiController`	Funzionale	Le funzionalità si concentrano sulla gestione dei contatti preferiti.
`InterfacciaAggiungiModificaController`	Procedurale	Gestisce il flusso di lavoro per aggiungere o modificare un contatto. Le operazioni includono la raccolta dei dati e il passaggio del contatto aggiornato alla rubrica.
`SalvaCaricaRubrica`	Funzionale	Lo scopo principale è gestire il salvataggio e il caricamento di dati (in formato JSON o CSV).
`SalvaCaricaPreferiti`	Funzionale	Simile a SalvaCaricaRubrica, ma specifico per i contatti preferiti.
`GestioneDuplicati`	Logica	Le operazioni effettuate sono simili, ma in contesti diversi.
`CampoNonValidoException`	Funzionale	Segnala errori relativi alla validazione dei campi, con metodi dedicati a costruire e restituire messaggi di errore.
`ContattoValidator`	Logica	Controlla la validità dei campi (nome, cognome, numeri di telefono, email) per garantire l'integrità dei dati.

Tabella 15: Analisi della Coesione dei Moduli del Sistema

Modulo	Livello di Accoppiamento	Descrizione
`Contatto`	Per Dati	Si è scelto un accoppiamento minimo per trasferire solo i dati necessari, garantendo che la classe resti focalizzata sulla rappresentazione del modello.
`Rubrica`	Per Timbro, Per Dati	Le sue dipendenze sono coerenti con il suo ruolo come gestore centrale dei dati. Tuttavia, alcuni metodi come `searchContact` dipendono da parametri esterni.
`InterfacciaUtenteController`	Per Timbro, Per Dati	L'accoppiamento con l'interfaccia GestioneRubrica è necessario per trasferire dati complessi, come le liste osservabili (rubriche) o dei numeri, tra l'interfaccia grafica e il modello, garantendo sincronizzazione dinamica e aggiornamenti in tempo reale.
`MenuPreferitiController`	Per Timbro, Per Dati	Le liste osservabili sono passate per facilitare la gestione dinamica dei dati preferiti, minimizzando la complessità di sincronizzazione con la rubrica principale.
`InterfacciaAggiungiModificaController`	Per Dati	Si è scelto un accoppiamento per dati essenziali per garantire una gestione diretta delle operazioni di inserimento e modifica.
`SalvaCaricaRubrica`	Per Timbro	L'accoppiamento per timbro è necessario per scambiare in modo sicuro e consistente liste osservabili tra la rubrica dei preferiti e i file JSON, sfruttando la libreria Jackson.
`SalvaCaricaPreferiti`	Per Timbro	L'accoppiamento è il medesimo di SalvaCaricaRubrica.
`GestioneDuplicati`	Per Timbro, Per Dati	L'accoppiamento consente di verificare duplicati direttamente sulle liste osservabili.
`CampoNonValidoException`	Per Dati	Un accoppiamento per dati è sufficiente per segnalare errori specifici.
`ContattoValidator`	Per Dati	L'accoppiamento consente un controllo rigoroso della validità dei dati, segnalando errori attraverso eccezioni, mantenendo la separazione dei compiti.

Tabella 16: Analisi dell'Accoppiamento dei Moduli del Sistema

11.0.1 Relazioni ed Ereditarietà

La classe **astratta** `VisualizzazioneContatti` centralizza la logica comune per la visualizzazione dei dettagli di un contatto ed è estesa da `MenuPreferitiController` e `InterfacciaUtenteController`. Questo approccio riduce la duplicazione del codice e semplifica la manutenzione.
L'interfaccia `GestioneRubricaController` definisce i metodi essenziali per la gestione dei contatti (`addAction`, `deleteAction`, `editAction`, `searchAction`) ed è implementata da entrambe le classi per garantire un contratto comune tra i moduli.
L'interfaccia `Gestione Rubrica` è stata introdotta in un'ottica di aggiornamento del codice. Se in futuro si volessero implementare altre rubriche, ad esempio con una persistenza su database o sincronizzata con un'API REST, o semplicemente modificare la logica di quella principale, sarà possibile farlo perché i controller non mantengono un riferimento diretto alla classe Rubrica.

11.0.2 Possibili Miglioramenti sull'Accoppiamento

Nonostante il design del sistema presenti già un buon livello di accoppiamento, ci sono alcune aree in cui possono essere apportati miglioramenti per ridurlo ulteriormente. I metodi di la ricerca, ad esempio, sfruttano un filtro passato come parametro. Si

potrebbe implementare una nuova interfaccia per separare la logica di filtraggio dalla classe principale Rubrica.

12 Principi di Progettazione Adottati

La progettazione della Rubrica è stata sviluppata seguendo principi di ingegneria del software che garantiscono la qualità, la manutenibilità e la scalabilità del sistema. Tali principi sono stati applicati per evitare il debito tecnico e per creare una base solida per future estensioni e aggiornamenti.

12.1 Refactoring

12.1.1 KISS ("Keep It Simple, Stupid!")

Sono state evitate complessità inutili e soluzioni troppo sofisticate. Ogni metodo è focalizzato su una singola responsabilità, riducendo al minimo il rischio di errori.
Ad esempio, nella classe `InterfacciaAggiungiModificaController` metodi come `handleValidationError` e `closeWindow` gestiscono azioni specifiche (gestione degli errori e chiusura della finestra) in modo separato, mantenendo il codice modulare e focalizzato. Così come l'inizializzazione e la raccolta dei dati dai campi di input (`collectNumbers` e `collectEmails`).

12.1.2 DRY ("Don't Repeat Yourself")

Il codice è progettato per evitare duplicazioni, centralizzando logiche comuni in metodi riutilizzabili.
Ad esempio, nella classe `Rubrica`, la logica per sincronizzare un contatto tra la lista dei contatti e quella dei preferiti è centralizzata nel metodo `synchronizeContacts`. Inoltre, la gestione del salvataggio dei contatti e dei preferiti è affidata al metodo `saveData()`, che invoca `saveFavorites()`. Infine, la verifica della corrispondenza di un contatto con i parametri di ricerca è gestita dal metodo `matchesContact`, e invocato su entrambe le rubriche.

12.1.3 Separazione delle Preoccupazioni e Ortogonalità

La logica di interazione con l'utente, la validazione dei e la gestione dei dati sono chiaramente distinte, ogni componente ha una responsabilità ben definita.
I controller come `MenuPreferitiController` e `InterfacciaAggiungiModificaController` si occupano esclusivamente dell'interazione con l'utente, senza contenere logiche complesse di gestione dei dati. La validazione dei contatti è affidata alle classi `ContattoValidator` e `GestioneDuplicati`, mentre le operazioni di salvataggio e caricamento dei dati sono delegate alle classi `SalvaCaricaRubrica` e `SalvaCaricaPreferiti`.

12.1.4 Principio della Minima Sorpresa

I nomi delle classi e dei metodi sono autoesplicativi garantendo che il comportamento del codice sia prevedibile e intuitivo per gli sviluppatori.

12.1.5 Regola del Boy-Scout

Ogni modifica al codice è stata accompagnata da un miglioramento, per garantire che il sistema sia sempre in condizioni migliori rispetto allo stato precedente.

13 Documentazione di Testing

Le classi di test sono state utilizzate per verificare e validare il corretto funzionamento del codice implementato. Ogni classe di test si concentra su una parte specifica del programma e contiene metodi che testano comportamenti e funzionalità.

Test	**Contatto**
TestItems	ContattoConstructor
Input	contact = new Contatto("Dati")
Oracle	Dato=contatto.getDato()

Test	**Contatto**
TestItems	testSetFav

Input	Contatto contatto = new Contatto("Dati");
	contatto1.setFav(true);
	contatto2.setFav(false);

Oracle	contatto1.isFav()); !contatto2.isFav()

Test	**Contatto**
TestItems	testCompareTo
Input	Contatto contatto1 = new Contatto(); Contatto contatto2 = new Contatto(); Contatto contatto3 = new Contatto(); Contatto contatto4 = new Contatto();

Oracle	contatto1.compareTo(contatto2) > 0;
	contatto2.compareTo(contatto1) < 0;
	contatto4 == (contatto3);

Test	**Contatto**
TestItems	testEquals
Input	Contatto contatto1 = new Contatto("Dati"); Contatto contatto2 = new Contatto("Dati"); Contatto contatto3 = new Contatto("Dati");
Oracle	contatto1 == contatto2; contatto1 != contatto3;

Test	**Contatto**
TestItems	testHashCode
Input	Contatto contatto1 = new Contatto("Dati"); Contatto contatto2 = new Contatto("Dati");
Oracle	contatto1.hashCode() == contatto2.hashCode();

Test	Rubrica
TestItems	AddContact_ValidContact_ShouldAddSuccessfully

Input	rubrica = new Rubrica();
	rubrica.setContactList(FXCollections.observableArrayList());
	rubrica.setFavoriteList(FXCollections.observableArrayList());
	rubrica.addContact(name, surname, numbers, emails, note, gen);

Oracle	contactList.size() == 1;
	addedContact.getName() == name;
	addedContact.getSurname() == surname;
	addedContact.getNumbers() == numbers;
	addedContact.getEmails() == emails;
	addedContact.getNote() == note;

Test	Rubrica
TestItems	testAddContact_DuplicateContact_ShouldThrowException

Input	rubrica = new Rubrica();
	rubrica.setContactList(FXCollections.observableArrayList());
	rubrica.setFavoriteList(FXCollections.observableArrayList());
	rubrica.addContact(name, surname, numbers, emails, note, "");

Oracle	CampoNonValidoException viene lanciata quando si tenta di aggiungere un contatto duplicato.

Test	Rubrica
TestItems	testEditContact_ValidEdit_ShouldUpdateContact

Input	rubrica = new Rubrica();
	rubrica.setContactList(FXCollections.observableArrayList());
	rubrica.setFavoriteList(FXCollections.observableArrayList());
	Contatto oldContact = new Contatto("Dati");
	rubrica.getContactList().add(oldContact);
	String newName = "";
	String newSurname = "";
	List<String> newNumbers = Arrays.asList("");
	List<String> newEmails = Arrays.asList("");
	String newNote = "";
	String newGen = "";
	rubrica.editContact(oldContact, newName, newSurname, newNumbers, newEmails, newNote, "newGen");

Oracle	contactList.size() == 1;
	updatedContact.getName() == newName;
	updatedContact.getSurname() == newSurname;
	updatedContact.getNumbers() == newNumbers;
	updatedContact.getEmails() == newEmails;
	updatedContact.getNote() == newNote;
	updatedContact.getNote() == newGen;

Test	Rubrica
TestItems	testDeleteContact_ExistingContact_ShouldRemoveContact

Input	rubrica = new Rubrica();
	rubrica.setContactList(FXCollections.observableArrayList());
	rubrica.setFavoriteList(FXCollections.observableArrayList());
	Contatto contact = new Contatto("Dati");
	rubrica.getContactList().add(contact);
	rubrica.deleteContact(contact);

Oracle	rubrica.getContactList().isEmpty()

Test	Rubrica
TestItems	testAddToFavorites_ValidContact_ShouldAddToFavorites

Input	rubrica = new Rubrica();
	rubrica.setContactList(FXCollections.observableArrayList());
	rubrica.setFavoriteList(FXCollections.observableArrayList());
	Contatto contact = new Contatto("Dati");
	rubrica.getContactList().add(contact);
	rubrica.addToFavorites(contact);

Oracle	favoriteList.size() == 1; contact.isFav()

Test	Rubrica
TestItems	testRemoveFromFavorites_ExistingFavorite_ShouldRemoveFromFavorites

Input	rubrica = new Rubrica();
	rubrica.setContactList(FXCollections.observableArrayList());
	rubrica.setFavoriteList(FXCollections.observableArrayList());
	Contatto contact = new Contatto("Dati");
	rubrica.getContactList().add(contact);
	rubrica.addToFavorites(contact);
	rubrica.removeFromFavorites(contact);

Oracle	favoriteList.isEmpty() !contact.isFav()

Test	Rubrica
TestItems	testAddToFavorites_ValidContact_ShouldAddToFavorites

Input	rubrica = new Rubrica();
	rubrica.setContactList(FXCollections.observableArrayList());
	rubrica.setFavoriteList(FXCollections.observableArrayList());
	Contatto contact = new Contatto("Dati");
	rubrica.getContactList().add(contact);
	rubrica.addToFavorites(contact);

Oracle	favoriteList.size() == 1; contact.isFav()

Test	Rubrica
TestItems	testRemoveFromFavorites_ExistingFavorite_ShouldRemoveFromFavorites

Input	rubrica = new Rubrica();
	rubrica.setContactList(FXCollections.observableArrayList());
	rubrica.setFavoriteList(FXCollections.observableArrayList());
	Contatto contact = new Contatto("Dati");
	rubrica.getContactList().add(contact);
	rubrica.addToFavorites(contact);
	rubrica.removeFromFavorites(contact);

Oracle	favoriteList.isEmpty() !contact.isFav()

Test	Rubrica
TestItems	testDeleteContact_ShouldRemoveAlsoFromFavorites

Input	rubrica = new Rubrica();
	rubrica.setContactList(FXCollections.observableArrayList());
	rubrica.setFavoriteList(FXCollections.observableArrayList());
	Contatto contact = new Contatto("Dati");
	rubrica.getContactList().add(contact);
	rubrica.addToFavorites(contact);
	rubrica.deleteContact(contact);

Oracle	contactList.isEmpty() favoriteList.isEmpty()

Test	SalvaCaricaPreferiti
TestItems	SalvaCaricaPreferiti.saveFavoritesAddressBook
Input	testFavorites = FXCollections.observableArrayList(new Contatto(), new Contatto()) SalvaCaricaPreferiti.saveFavoritesAddressBook(testFavorites) ObservableList<Contatto> loadedFavorites = SalvaCaricaPreferiti.loadFavoritesAddressBook()
Oracle	for i in [0, SIZE], testFavorites.get(i).getDati == loadedFavorites.get(i).getDati

Test	SalvaCaricaRubrica
TestItems	SalvaCaricaRubrica.SaveAddressBook
Input	testContacts = FXCollections.observableArrayList(new Contatto(), new Contatto()) SalvaCaricaRubrica.saveAddressBook(testContacts) ObservableList<Contatto> loadAddressBook = SalvaCaricaPreferiti.loadAddressBook()
Oracle	for i in [0, SIZE], testContacts.get(i).getDati == loadAddressBook.get(i).getDati

Test	SalvaCaricaRubrica
TestItems	SalvaCaricaRubrica.LoadAddressBook
Input	testContacts = FXCollections.observableArrayList(new Contatto(), new Contatto()) SalvaCaricaRubrica.saveAddressBook(testContacts) ObservableList<Contatto> loadedContact = SalvaCaricaRubrica.loadAddressBook()
Oracle	loadedContact!=0 loadedContact.size == testContacts.size() for i in [0, SIZE], testContacts.get(i).getDati == loadedContact.get(i).getDati

Test	SalvaCaricaRubrica
TestItems	SalvaCaricaRubrica.exportToCSV
Input	testContacts = FXCollections.observableArrayList(new Contatto(), new Contatto()) SalvaCaricaRubrica.exportToCSV(testContacts) List<String> lines = Files.readAllLines(Paths.get(TEST_FILE_CSV))
Oracle	Files.exist(Paths.get(TEST_FILE_CSV)) loadedContact.size == testContacts.size() DatiInseriti=DatiRilevati

Test	ContattoValidator
TestItems	ValidateFields.exportToCSV
Input	validContact = new Contatto() invalidContact = new Contatto()
Oracle	!Throw -> ContattoValidator.validateFields(validContact.getFields()) Throw -> ContattoValidator.validateFields(invalidContact.getFields())

Test	GestioneDuplicati
TestItems	IsAddValid
Input	validContact = new Contatto() Sono stati creati 8 contatti: **Caso 1**: Un contatto iniziale di riferimento **Caso 2**: Contatto con numero di telefono duplicato **Caso 3**: Contatto con email duplicata **Caso 4**: Contatto con nome e cognome duplicati **Caso 5**: Contatto con numeri e email univoci (nessun duplicato) **Caso 6**: Contatto con cognome duplicato, ma nome diverso (valido) **Caso 7**: Contatto con numeri parzialmente duplicati **Caso 8**: Contatto completamente diverso (nessun duplicato) ObservableList<Contatto> contactList = FXCollections.observableArrayList()
Oracle	!isAddValid(contatto2, contactList) !isAddValid(contatto3, contactList) !isAddValid(contatto4, contactList) isAddValid(contatto5, contactList) isAddValid(contatto6, contactList) isAddValid(contatto7, contactList) isAddValid(contatto8, contactList)

Test	`GestioneDuplicati`
TestItems	`IsModifyValid`

Input	`oldContact = new Contatto()`
	`testContact = new Contatto()`
	`ObservableList<Contatto> contactList = FXCollections.observableArrayList()`
	Sono stati modificati 7 contatti:
	Caso 1: Contatto aggiornato con nome e cognome duplicati
	Caso 2: Contatto aggiornato con numero duplicato
	Caso 3: Contatto aggiornato con email duplicata
	Caso 4: Contatto aggiornato valido (nessun duplicato)
	Caso 5: Contatto aggiornato con cognome uguale (valido)
	Caso 6: Contatto aggiornato con numero e email del contatto originale (valido)
	Caso 7: Contatto aggiornato con numero e email parzialmente duplicati

Oracle	`!isModifyValid(oldContact, contatto1, contactList)`
	`!isModifyValid(oldContact, contatto2, contactList)`
	`!isModifyValid(oldContact, contatto3, contactList)`
	`isModifyValid(oldContact, contatto4, contactList)`
	`isModifyValid(oldContact, contatto5, contactList)`
	`isModifyValid(oldContact, contatto6, contactList)`
	`!isModifyValid(oldContact, contatto7, contactList)`